Nama : Firman Ramadhan
Kelas : 2IA11
NPM : 50422584
Mata Kuliah : Pemrograman Berbasis Objek

Rangkuman Bab 1-6

I. Pengenalan Pemrograman Berbasis Objek

Pemrograman Berorientasi Objek (OOP) adalah paradigma pemrograman yang fokus pada objek sebagai metode untuk menyelesaikan masalah program yang kompleks. Objek adalah entitas dengan atribut, perilaku, dan kadang-kadang kondisi. OOP berkembang dari program terstruktur pada tahun 1960, terutama dari bahasa C dan Pascal, memungkinkan penulisan program yang sulit menjadi lebih mudah. OOP tidak hanya mempertimbangkan bagaimana menyelesaikan masalah secara terstruktur, tetapi juga menitikberatkan pada objek-objek yang dapat menyelesaikan masalah tersebut. OOP berfungsi baik dengan pendekatan Analisis dan Desain Berorientasi Objek (OOAD).

II. Perbedaan Pemrograman Berbasis Objek dan Pemrograman Terstruktur

Secara umum, OOP menyediakan pendekatan yang lebih terstruktur dan modular dalam membangun program dengan menekankan penggunaan objek sebagai unit dasar, sementara pemrograman terstruktur cenderung lebih sederhana dan langsung dalam aliran eksekusinya.

Pendekatan Dasar:

- Pemrograman Berbasis Objek (OOP): Memodelkan program sebagai kumpulan objek yang saling berinteraksi untuk menyelesaikan masalah.
- Pemrograman Terstruktur: Memodelkan program sebagai serangkaian instruksi yang dijalankan secara berurutan untuk mencapai tujuan tertentu. Struktur program biasanya terdiri dari fungsi atau prosedur yang memanipulasi data.

Fokus Utama:

- OOP : Fokus pada objek-objek sebagai unit utama dalam program, yang mewakili entitas atau konsep dalam dunia nyata.
- Pemrograman Terstruktur : Fokus pada alur eksekusi program yang terorganisir berdasarkan urutan instruksi.

Pengorganisasian Data:

- OOP: Data dan metode yang beroperasi pada data tersebut dibungkus bersama dalam objek. Ini disebut enkapsulasi.
- Pemrograman Terstruktur: Data seringkali diorganisir dalam struktur data seperti array atau struktur, dan metode memanipulasi data tersebut dapat berada di luar struktur data.

Polimorfisme:

- OOP: Memungkinkan metode dengan nama yang sama berperilaku berbeda di berbagai kelas objek.
- Pemrograman Terstruktur: Polimorfisme tidak secara eksplisit didukung; fungsi atau prosedur biasanya memiliki nama yang unik.

III. Class Pada PBO

Class adalah cetak biru atau blueprint dari object. Class digunakan hanya untuk membuat kerangka dasar.
Sebagai analogi, class bisa diibaratkan dengan kendaraan. Kita tahu bahwa kendaraan memiliki ciri-ciri seperti merk, memiliki mesin, memiliki ban, dan beberapa ciri khas lain yang menyatakan sebuah benda tersebut adalah kendaraan. Selain memiliki ciri-ciri, sebuah kendaraan juga bisa dikenakan tindakan, seperti: menghidupkan mesin atau mematikan mesin.
Contoh Program:

```
// Deklarasi Class
public class Kendaraan{
    // Ini Adalah Class, semua konstruktor, variabel, method berada disini
}
```

IV. Objek Pada PBO

Objek dalam pemrograman berbasis objek adalah representasi dari entitas dalam dunia nyata, seperti orang, mobil, atau rekening bank. Objek memiliki atribut (state) dan perilaku (behavior). Misalnya, objek sepeda memiliki atribut seperti pedal, roda, dan warna, serta perilaku seperti menaikkan kecepatan dan perpindahan gigi.

Dalam pengembangan perangkat lunak berorientasi objek, objek menyimpan state-nya dalam variabel dan perilaku dalam metode atau fungsi. Ini memungkinkan representasi yang lebih abstrak dan modular dalam membangun program.

Untuk membuat objek, bisa menggunakan kata kunci new yang digunakan untuk membuat objek baru, selanjutnya menentukan 3 langkah untuk membuat sebuah objek. Yaitu: mendeklarasikan variable, membuat objek baru (Instansiasi) dan pemanggilan konstruktor.
Contoh Program:

```
public class Kendaraan{

    // Konstruktor Dengan Parameter
    public Kendaraan(String nama){
        System.out.println("Nama Kendaraannya Adalah "+ nama);
    }

    public static void main(String[]args){

        // Perintah untuk membuat objek jenis
        Kendaraan jenis = new Kendaraan("Pesawat Terbang");

    }

}

// Output = Nama Kendaraannya Adalah Pesawat Terbang
```

V. Pengertian dasar inheritance

Pewarisan adalah penggunaan kembali kelas yang ada untuk membuat kelas baru dengan menambahkan fitur baru. Setiap kelas turunan dapat menjadi induk bagi kelas turunan berikutnya. Constructor tidak diwariskan secara otomatis dalam pewarisan, kecuali menggunakan perintah "super". Java hanya mendukung pewarisan tunggal. Kelas yang memberikan warisan disebut superclass, sementara yang menerima disebut subclass. Untuk menerapkan inheritance dalam Java, gunakan "extends".

Contoh untuk memanggil constructor milik superclass-nya :

```
super()
super(parameter)
```

Contoh untuk memanggil atribut dan method milik superclass-nya :

```
super.namaAtribut
super.namaMethod(parameter)
```

VI. Pengertian dasar Encapsulation

Enkapsulasi (encapsulation) merupakan cara untuk melindungi property (atribut) / method tertentu dari sebuah kelas agar tidak sembarangan diakses dan dimodifikasi oleh suatu bagian program.

Accessors :

Cara untuk melindungi data yaitu dengan menggunakan access modifiers (hak akses). Ada 4 hak akses yang tersedia, yaitu default, public, protected, private.

No	Modifier	Pada class dan interface	Pada method dan variabel
1	Default (tidak ada modifier)	Dapat diakses oleh yang sepaket	Diwarisi oleh subkelas dipaket yang sama, dapat diakses oleh method-method yang sepaket
2	Public	Dapat diakses dimanapun	Diwarisi oleh subkelasnya, dapat diakses dimanapun
3	Protected	Tidak bisa diterapkan	Diwarisi oleh subkelasnya, dapat diakses oleh method-method yang sepaket
4	private	Tidak bisa diterapkan	Tidak dapat dikases dimanapun kecuali oleh method-method yang ada dalam kelas itu sendiri

Aksesabilitas	public	private	protected	default
Dari kelas yang sama	Ya	Ya	Ya	Ya
Dari sembarang kelas dalam paket yang sama	Ya	Tidak	Ya	Ya
Dari sembarang kelas di luar paket	Ya	Tidak	Tidak	Tidak
Dari subkelas dalam paket yang sama	Ya	Tidak	Ya	Ya
Dari subkelas di luar paket	Ya	Tidak	Ya	Tidak

VII. Polymorphism

- Overloading adalah diperbolehkannya dalam sebuah class memiliki lebih dari satu nama function/method yang sama tetapi memiliki parameter/argument yang berbeda.

```
public class Overloading {
    public void Tampil(){
        System.out.println("I love Java");
    }
    public void Tampil(int i){
        System.out.println("Method dengan 1 parameter = "+i);
    }
    public void Tampil(int i, int j){
        System.out.println("Method dengan 2 parameter = "+i+" & "+j);
    }
    public void Tampil(String str){
        System.out.println(str);
    }

    public static void main(String a[]){
        Overloading objek = new Overloading();
        objek.Tampil();
        objek.Tampil(8);
        objek.Tampil(6,7);
        objek.Tampil("Hello world");
    }
}
```

Overriding method adalah kemampuan dari subclass untuk memodifikasi method dari superclass-nya, yaitu dengan cara menumpuk (mendefinisikan kembali) method superclass-nya. Contoh overriding method dapat dilihat pada subclass “Mobil” yang mendefinisikan kembali method keterangan() dan hapus() dari class “Kendaraan”.

VIII. Inheritance

Untuk setiap burung, ada satu set properti yang telah ditetapkan yang umum untuk semua burung dan ada satu set properti yang khusus untuk burung tertentu. Oleh karena itu, secara intuitif, kita dapat mengatakan bahwa semua burung mewarisi ciri-ciri umum seperti sayap, kaki, mata, dll. Oleh karena itu, dalam cara berorientasi objek untuk merepresentasikan burung, pertama-tama kita mendeklarasikan kelas “Bird” dengan seperangkat properti yang umum untuk semua burung. Dengan melakukan ini, kita dapat menghindari menyatakan sifat-sifat umum ini di setiap burung yang kita buat. Sebagai gantinya, kita cukup mewariskan kelas “Bird” di semua burung yang kita buat.

```
// Implementing the bird class
public class Bird {

    // Few properties which
    // define the bird
    String animal = "All Birds";
    String color;
    int legs;

    // Few operations which the
    // bird performs
    public void eat()
    {
        System.out.println(
            "This bird has eaten");
    }

    public void fly()
    {
        System.out.println(
            "This bird is flying");
    }
}
```

Setelah kelas “Bird” dibuat, jika kita ingin membuat kelas “Pigeon”, maka kita cukup mewarisi kelas “Bird” di atas.

```java
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flys!!!");
    }
}
```

Dan jika kita ingin membuat method yang menggunakan atribut dari kelas ”Bird”, maka kita dapat memanggilnya dengan menggunakan keyword “super”.

```java
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flys!!!");
    }

    //super & this
    public void about(){
        System.out.println(super.animal+" live freely in the sky");
        System.out.println(this.animal+ " can recognise each letter of the human alphabet");
    }
}
```

IX. Encapsulation

Sekarang, kita telah mendefinisikan properti dari kelas “Bird” dan atribut yang dimiliki burung seperti warna, sayap, kaki dapat diinisialisasi dengan membuat objek kelas burung. Namun, jika kita hanya dapat mengubah properti dari kelas “Bird” hanya dengan referensi dari objek, maka atribut kehilangan informasi yang awalnya diinisialisasi.

Misalnya, katakanlah kita awalnya membuat merpati dengan warna abu-abu dengan membuat konstruktor, setiap pengguna dengan instance objek merpati dapat mengubah warna ini menjadi merah atau hitam hanya dengan merujuk atribut dengan kata kunci “this". Untuk menghindari hal ini, kita menyertakan properti ke dalam sebuah metode. Metode ini disebut getter dan setter atribut. Idenya adalah untuk hanya menyertakan inisialisasi dan pengambilan atribut dalam suatu metode alih-alih langsung merujuk atribut secara langsung. Ini juga memberikan keuntungan karena setter memberi kita kendali penuh dalam menetapkan nilai ke atribut dan membantu kita membatasi perubahan yang tidak perlu.

```java
// Implementing the bird class
public class Bird {

    // Few properties which
    // define the bird
    String animal = "All Birds";
    String color;
    int legs;

    // Implementing the getters and
    // setters for the color and legs.

    public void setColor(String color)
    {
        this.color = color;
    }

    public String getColor()
    {
        return this.color;
    }

    public void setLegs(int legs)
    {
        this.legs = legs;
    }

    public int getLegs()
    {
        return this.legs;
    }

    // Few operations which the
    // bird performs
    public void eat()
    {
        System.out.println(
            "This bird has eaten");
    }

    public void fly()
    {
        System.out.println(
            "This bird is flying");
    }
}
```

X. Polymorphism

Sekarang, kita ingin membuat merpati dapat terbang dengan menggunakan method “fly” namun kita ingin mendefinisikan tujuan terbang merpati tersebut dan kita ingin mendefinisikan makanan yang dimakan oleh merpati tersebut. Kita dapat melakukannya dengan menggunakan konsep overload method.

```java
// Creating the Pigeon class which
// extends the bird class
public class Pigeon extends Bird {

    String animal ="Pigeon";
    // Overriding the fly method
    // which makes this pigeon fly
    @Override
    public void fly() //override method
    {
        System.out.println(
            "Pigeon flys!!!");
    }

    //overload method to method in this class
    public void fly(String place){
        System.out.println("Pegeon flys to "+ place);
    }

    //overload method to method in parent class
    public void eat(String food)
    {
        System.out.println(
            "Pigeon eats "+food);
    }

    //super & this
    public void about(){
        System.out.println(super.animal+" live freely in the sky");
        System.out.println(this.animal+ " can recognise each letter of the human alphabet");
    }
}
```

XI. Testing

Setelah kita selesai membuat kelas “Bird” dan kelas “Pigeon”, kita perlu membuat sebuat kelas untuk menjalankan atau memanggil kelas-kelas tersebut beserta method-methodnya.

```
//encapsulation, inheritance, & polymorphism testing
public class OOPTest {
    public static void main(String args[]){
        Bird bird = new Bird();
        Pigeon pigeon = new Pigeon();

        bird.eat();
        bird.fly();

        System.out.println("");

        //polymorphism
        pigeon.eat("Seed");
        pigeon.fly();
        pigeon.fly("Sky");

        System.out.println("");

        //Mutators & inheritance
        pigeon.setColor("grey");
        pigeon.setLegs(2);
        System.out.println(
                "this pigeon has "+pigeon.getColor()+" color "
                + "and "+ pigeon.getLegs()+" legs");
        pigeon.eat();

        System.out.println("");
        pigeon.about();

    }
}
```